Digital Design LU

Lab Exercise 3

Jakob Lechner, Thomas Polzer
{lechner, tpolzer}@ecs.tuwien.ac.at
Department of Computer Engineering
University of Technology Vienna

Vienna, October 27, 2011

1 Required Reading

- Design flow tutorial
- VHDL modeling slides
- State machine slides
- Exercise 3 description (this document)

2 Task Description

Your task is to implement a serial port. The details are described in the following sections.

2.1 Interface

The generics of the serial port interface are described in Table 1 and the signals in Table 2.

Name	Functionality
CLK_FREQ	Actual clock frequency of the *clk* signal given in Hz.
$BAUD\_RATE$	The baud rate used for the serial port.
$SYNC\_STAGES$	Number of stages used in the input synchronizer.
TX_FIFO_DEPTH	Number of elements which can be stored within the transmitter FIFO.
RX_FIFO_DEPTH	Number of elements which can be stored within the receiver FIFO.

Table 1: Serial port generics description.

2.2 Interface Protocol

The interface protocol of the serial port is the same as for the FIFO buffers described in the IP core documentation. The transmitter port uses the write port of the FIFO, while the receiver port uses its read port.

The tx and rx signals use the standard UART protocol with 8 data bits, 1 stop bits and no parity. The used BAUD rate is configured using the corresponding generics.

Name	Direction	Signal width	Functionality
clk	in	1	Global clock signal.
res_n	in	1	Global reset signal (low active, not internally synchronized).
tx_data	in	8	Data which should be transmitted to the host.
tx_wr	in	1	If 1, a new data byte is present on *tx_data* and should be copied to the internal buffer.
tx_free	out	1	If 1, the internal buffer is not full and a new data byte may be written into the buffer.
rx_data	out	8	The byte which was last read from the receiver FIFO.
rx_rd	in	1	If 1, the next byte is read from the receiver FIFO if it is available. The byte is available at the *rx_data* port. If no byte is available, the value of the *rx_data* port is undefined.
rx_data_full	out	1	If 1, the receiver FIFO buffer is full and characters may have been lost because they could not be stored in the buffer.
rx_data_empty	out	1	If 1, the receiver FIFO is empty, otherwise received bytes are available in the buffer.
rx	in	1	The receive signal of the UART (Host → Device).
tx	out	1	The transmit signal of the UART (Device → Host).

Table 2: Serial port signal description.

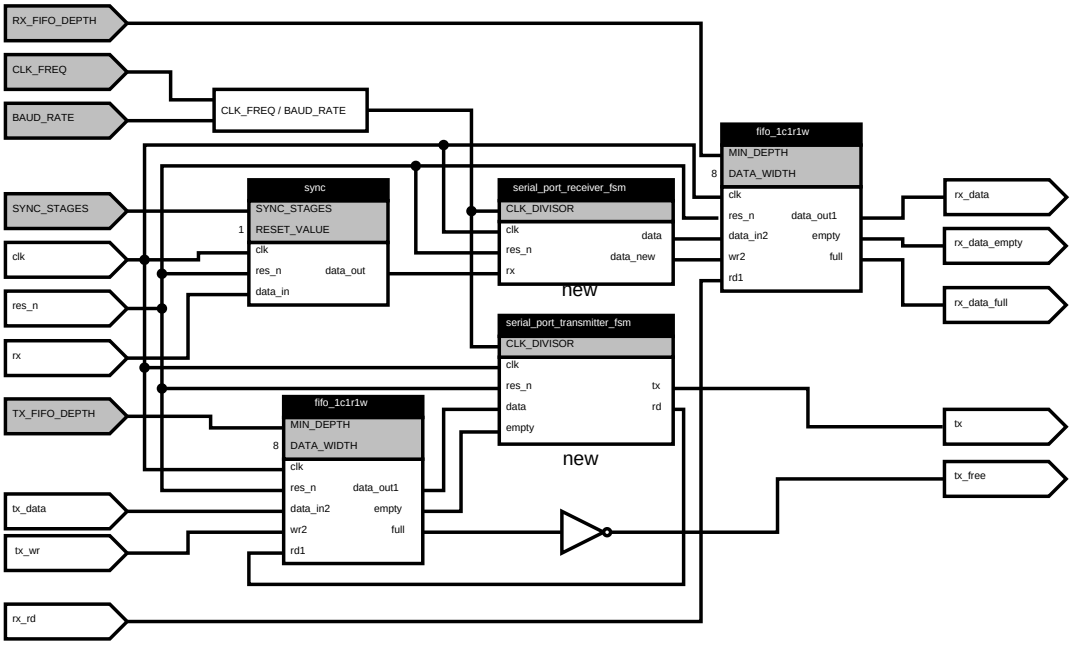

Figure 1: Serial port internal circuitry.

2.3 Internal Structure

The serial port consists of five components, namely a synchronizer, two FIFO buffers and the two state machines which implement the receiver and the transmitter. Figure 1 shows how these components are connected together and how the input and output signals are assigned.

2.4 State machine Description

The serial port is implemented using two state machines, namely the receiver and transmitter FSM. The operation of the receiver FSM is described by its state chart (see Figure 2), its state transition table (see Table 3) and its output behavior (see Table 4).

The operation of the transmitter FSM is described by its state chart (see Figure 3), its state transition table (see Table 5) and its output behavior (see Table 6).

2.5 Implementation

Start by implementing the two state machines independently. The implementation must follow the three process method. Create behavioral simulations to validate the correctness of your implementation.

State	Condition	Next state
$IDLE$	$rx = 1$	$WAIT\_START\_BIT$
$WAIT\_START\_BIT$	$rx = 0$	$GOTO\_MIDDLE\_OF\_START\_BIT$
$GOTO\_MIDDLE\_OF\_START\_BIT$	$clk\_cnt = \frac{CLK\_DIVISOR}{2} - 2$	$MIDDLE\_OF\_START\_BIT$
$MIDDLE\_OF\_START\_BIT$	always	$WAIT\_DATA\_BIT$
$WAIT\_DATA\_BIT$	$clk\_cnt = CLK\_DIVISOR - 2$	$MIDDLE\_OF\_DATA\_BIT$
$MIDDLE\_OF\_DATA\_BIT$	$bit\_cnt < 7$	$WAIT\_DATA\_BIT$
	$bit\_cnt = 7$	$WAIT\_STOP\_BIT$
$WAIT\_STOP\_BIT$	$clk\_cnt = CLK\_DIVISOR - 2$	$MIDDLE\_OF\_STOP\_BIT$
$MIDDLE\_OF\_STOP\_BIT$	$rx = 0$	$IDLE$
	$rx = 1$	$WAIT\_START\_BIT$

Table 3: Serial port receiver FSM state transition table.

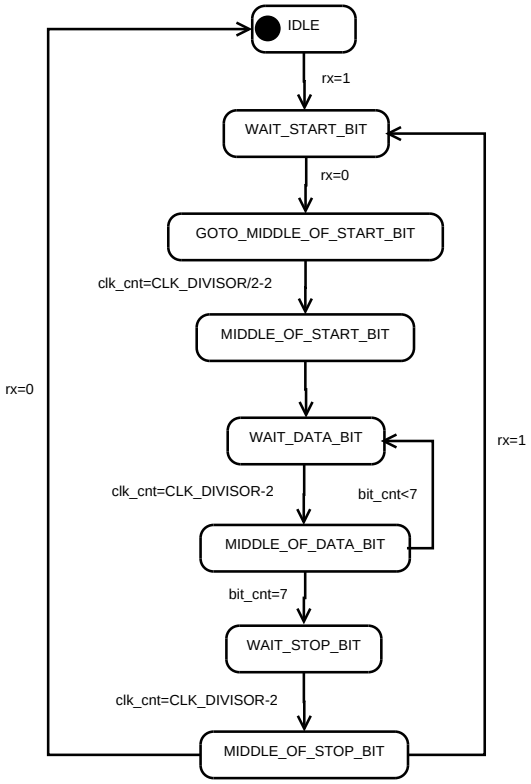

Figure 2: Serial port receiver state machine.

Afterwards create a structural VHDL design which connects all components (synchronizer, FIFO buffers and the state machines) corresponding to Figure 1. You should also use a behavioral simulation to validate its functionality.

2.6 Integration into your Exercise 2 Design

To demonstrate the correct functionality of your serial port design when really implemented in hardware, add the serial port component to your Exercise 2 design. Therefore the receiver part of your serial port should be connected to the output logic, while the transmitter part should be used to transmit the ASCII codes entered using the keypad to the PC (output of the PS/2 to ASCII converter). For simplicity reasons the tx_free port is left unconnected.

For the rx and tx signals you have to create new ports in your top level entity and map them to the corresponding FPGA pins.

State	Outputs
Default	clk_cnt keeps its data bit_cnt keeps its data $data\_int$ keeps its data $data\_new = 0$ $data\_out$ keeps its data
$IDLE$	
$WAIT\_START\_BIT$	$bit\_cnt = 0$ $clk\_cnt = 0$
$GOTO\_MIDDLE\_OF\_START\_BIT$	$clk\_cnt = clk\_cnt + 1$
$MIDDLE\_OF\_START\_BIT$	$clk\_cnt = 0$
$WAIT\_DATA\_BIT$	$clk\_cnt = clk\_cnt + 1$
$MIDDLE\_OF\_DATA\_BIT$	$clk\_cnt = 0$ $bit\_cnt = bit\_cnt + 1$ $data\_int =$ $rx\&data\_int(7\ downto\ 1)$
$WAIT\_STOP\_BIT$	$clk\_cnt = clk\_cnt + 1$
$MIDDLE\_OF\_STOP\_BIT$	$data\_new = 1$ $data\_out = data\_int$

Table 4: Serial port receiver FSM output table.

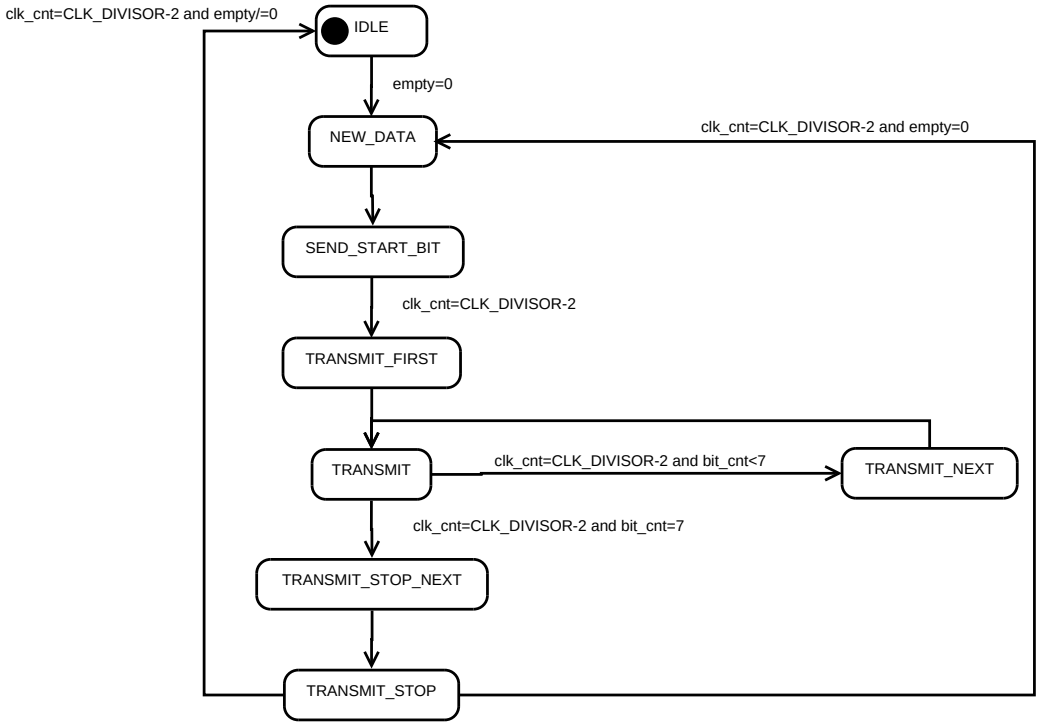

Figure 3: Serial port transmitter state machine.

State	Condition	Next state
$IDLE$	$empty = 0$	NEW_DATA
NEW_DATA	always	$SEND\_START\_BIT$
$SEND\_START\_BIT$	$clk\_cnt = CLK\_DIVISOR - 2$	$TRANSMIT\_FIRST$
$TRANSMIT\_FIRST$	always	$TRANSMIT$
$TRANSMIT\_NEXT$	always	$TRANSMIT$
$TRANSMIT$	$clk\_cnt = CLK\_DIVISOR - 2$ and $bit\_cnt = 7$	$TRANSMIT\_STOP\_NEXT$
	$clk\_cnt = CLK\_DIVISOR - 2$ and $bit\_cnt < 7$	$TRANSMIT\_NEXT$
$TRANSMIT\_STOP\_NEXT$	always	$TRANSMIT\_STOP$
$TRANSMIT\_STOP$	$clk\_cnt = CLK\_DIVISOR-2$ and $empty = 0$	NEW_DATA
	$clk\_cnt = CLK\_DIVISOR-2$ and $empty \neq 0$	$IDLE$

Table 5: Serial port transmitter FSM state transition table.

State	Outputs
Default	clk_cnt keeps its data bit_cnt keeps its data $transmit\_data$ keeps its data $rd = 0$ $tx = 1$
$IDLE$	
NEW_DATA	$rd = 1$ $clk\_cnt = 0$
$SEND\_START\_BIT$	$clk\_cnt = clk\_cnt + 1$ $tx = 0$
$TRANSMIT\_FIRST$	$clk\_cnt = 0$ $bit\_cnt = 0$ $tx = 0$ $transmit\_data = data$
$TRANSMIT\_NEXT$	$clk\_cnt = 0$ $bit\_cnt = bit\_cnt + 1$ $tx = transmit\_data(0)$ $transmit\_data(6\ downto\ 0) =$ $transmit\_data(7\ downto\ 1)$
$TRANSMIT$	$clk\_cnt = clk\_cnt + 1$ $tx = transmit\_data(0)$
$TRANSMIT\_STOP\_NEXT$	$clk\_cnt = 0$ $tx = transmit\_data(0)$
$TRANSMIT\_STOP$	$clk\_cnt = clk\_cnt + 1$

Table 6: Serial port transmitter FSM output table.

3 Submission Specification

The results are again handed in via myTI. The deadline is November 11^{th}, 2011, 23:59. Upload a ZIP file containing the following information:

- Your lab protocol as PDF
- The complete VHDL source code of the system (including your serial port and the PLL!)
- The SDC file containing the clock definition
- Your Quartus project (don't forget a cleanup!) or a TCL script creating the project (including the pin mappings!)

Make sure the submitted Quartus project is compilable. All submissions which can not be compiled will be graded with zero points!